3.14 Automation Plug-in

3.14.1 Description

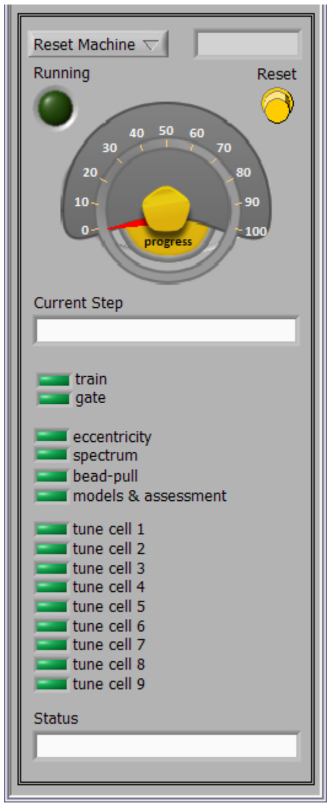
The automation plug-in executes automation scripts. Each script interacts with plug-ins and performs a desired sequence of operations. Currently, there are three scripts that can be selected for execution:

- Measure & Tune – performs all the measurements and the cavity tuning
- Measure Only – performs all the measurements
- Tune Only – performs the cavity tuning
- Reset Machine – resets the system to a known, safe state
- User Script – script developed by the user

As with any other plug-in the execution of a script can be suspended or aborted using the same buttons as for other plug-ins. The automation plug-in is selected by the switch on the control window that toggles between the Manual and Automatic modes.

3.14.2 User Interface

The user interface of the automation plug-in is shown below.



The selector located at the top of the window is used to choose a script. Next to it is the elapsed time display. The Running indicator flashes green when the script is running and red upon errors.

The progress indicator advances and reaches 100 % when the script is normally completed. The step being executed is shown directly below that indicator. A set of fifteen binary indicators show the completion of various major tuning tasks. Finally, Status shows messages displayed by the script. The Reset button is used to clear the display of the plug-in.

3.14.3 Automation scripts

The files that contain automation scripts have the extension **tas**, for **t**uning **a**utomation **s**cript. Automation scripts are sequences of instructions interpreted by the automation plug-in. The script is interpreted until the END instruction is encountered, which terminates the execution. The syntax of all script instructions is the same:

label command plug-in parameter jump-label indicator time-out

The instruction elements have the following meaning:

- **label** is a positive number identifying the command location in the script
- **command** is one of the implemented automation commands
- **plug-in** is the name of a plug-in
- **parameter** is a command-specific value
- **jump-label** is the label where the control will be transferred next (used only by the branch, jump and subroutine call commands)
- **indicator** is a positive number [0-14] identifying one of the automation plug-in front panel indicators
- **time-out** is a positive integer value in seconds (effective only for the WAIT and WAIT-ANY commands).

3.14.4 Command Return Value

Some commands also return a return value. The return value is a string specific to a particular command. The following are all examples of valid return values: OK, YES, NO, UNKNOWN, YELOW, etc.

Let us examine the following instruction:

345 WAIT gate Done 0 1 240

This instruction has a label of 345 and will cause the script to wait for the plug-in Gate to send the Done reply. The indicator number 1 will change color to yellow (busy) and keep this color until the awaited reply is received or time-out occurs. The time-out is set to 240 seconds. If the correct reply is received, the indicator will change its color to green. Upon time-out the indicator will change its color to red. If the indicator number is set to a negative value, such as -1, no indicator is used.

3.14.5 Comments

Comments are any text that follows the semicolon (;) character. Comments can be placed alone on the line or can be placed at the end of the instruction, as shown below.

```
; This section of code demonstrates the use of comments
10  DELAY –  10     0 0 0    ; delay execution for 10 seconds
; ………………………..
```

3.14.6 Abort Handler

The user can pause or abort the execution of a script. When the user decides to abort the script a special abort sequence of instructions, called an **abort handler**, is executed. This sequence always starts from the instruction with the **label 9999**. An abort handler is a mandatory part of any script.

3.14.7 Script Commands

Here is a list of all implemented script commands:

- **ALERT** – sound an alert signal
  SYNTAX: *label ALERT - - 0 0 0*
  EXAMPLE: *100 ALERT - - 0 0 0*
  RETURN: none

- **BEQ** – branch if the last return value is equal patameter
  SYNTAX: *label BEQ - parameter jump-label 0 0*
  EXAMPLE: *10 BEQ - ERROR 125 0 0 ; jump to 125 if the last returned value is ERROR*
  RETURN: none

- **BNEQ** – branch if the last return value is not equal parameter
  SYNTAX: *label BNEQ - parameter jump-label 0 0*
  EXAMPLE: *10 BNEQ - ERROR 125 0 0 ; jump to 125 if the last returned value is not ERROR*
  RETURN: none

- **CAMERA** – get the camera status and put it in the return value
  SYNTAX: *label CAMERA - - 0 0 0*
  EXAMPLE: *10 CAMERA - - 0 0 0*
  RETURN: OK, Unknown, Readout Error, Saturation, Low Intensity, Out of Limit, Timeout

- **CELLPROMPT** – prompt the user whether to tune the given cell and show tuning parameters
  SYNTAX: *label CELLPROMPT - [1-9] 0 0 0*
  EXAMPLE: *100 CELLPROMPT - 3 0 0 0*
  RETURN: YES, NO

- **CODE** – set the return code
  SYNTAX: *label CODE - return_code 0 0 0*
  EXAMPLE: *100 CODE - ERR 0 0 0*
  RETURN: none

- **DELAY** – delay (pause) the execution for a specified number of seconds
  SYNTAX: *label DELAY - seconds 0 0 0*
  EXAMPLE: *10 DELAY - 5 0 0 0 ; delay 5 seconds*
  RETURN: none

- **DISABLE** – disable the controls in the plug-in UI
  SYNTAX: *label DISABLE plug-in - 0 0 0*
  EXAMPLE: *100 DISABLE gate - 0 0 0*
  RETURN: none

- **DOVETAIL** – return whether the dovetail configuration is OK to close the gate
  SYNTAX: *label DOVETAIL - - 0 0 0*
  EXAMPLE: *100 DOVETAIL - - 0 0 0*
  RETURN: YES, NO

- **ENABLE** – enable the controls in the plug-in UI
  SYNTAX: *label ENABLE plug-in - 0 0 0*
  EXAMPLE: *100 ENABLE gate - 0 0 0*
  RETURN: none

- **END** – terminate the execution and show text in the message field
  SYNTAX: *label END - text 0 0 0*
  EXAMPLE: *9000 END - Script\ssucceded 0 0 0 ; successful completion*
  RETURN: none

- **ERROR** – log an error
  SYNTAX: *label ERROR - error_description 0 0 0*
  EXAMPLE: *9100 ERROR - Tuning\sscript\serror:\s$$ 0 0 0*
  RETURN: none

- **FLATNESS** – check field flatness calculated by the bead-pull measurement
  SYNTAX: *label FLATNESS - - 0 0 0*
  EXAMPLE: *100 FLATNESS - - 0 0 0*
  RETURN: OK if flatness >= 20 %
  ERROR if getting flatness failed
  Otherwise, field flatness in percent

- **GOSUB** – invoke the subroutine located at a given label (sub-label) and pass it a parameter. The parameter can be retrieved in the subroutine using $$
  SYNTAX: *label GOSUB - param sub-label 0 0*
  RETURN: none
  EXAMPLE: *10 GOSUB - GREEN 300 0 0 ; call subroutine with GREEN as a parameter*
  ……
  ; Swtch light subroutine
  300 LIGHT - $$ 0 0 0 ; switch requested light
  0 RETURN - - 0 0 0 ; return to caller

- **INDICATOR** – set the color or state of the selected indicator
  The allowed values are: ON, OFF, YELLOW, RED, GREEN
  SYNTAX: *label INDICATOR - value 0 indicator 0*
  EXAMPLE: *100 INDICATOR - YELLOW 0 5 0*
  RETURN: none

- **INFO** – log a message
  SYNTAX: *label INFO - message_text 0 0 0*
  EXAMPLE: *100 INFO - Tuning\scomplited 0 0 0*
  RETURN: none

- **INIT** – initialize plug-in (send the INIT command to plug-in) and flash the selected indicator
  SYNTAX: *label INIT plug-in - 0 indicator 0*
  EXAMPLE: *10 INIT gate - 0 1 0 ; initialize gate*
  RETURN: none

- **INTERLOCK** – check the interlock status and return the active interlock name (MOTION, DAQ, ETS) or OK. Return ERROR, when interlock subsystem failed.
  SYNTAX: *label INTERLOCK - - 0 0 0*
  EXAMPLE: *10 INTERLOCK - - 0 0 0 ; check interlocks*
  RETURN: OK, MOTION, DAQ, ETS, ERROR

- **JMP** – unconditional jump to the specified label
  SYNTAX: *label JMP - - jump-label 0 0*
  EXAMPLE: *10 JMP - - 125 0 0 ; jump to 125*
  RETURN: none

- **LIGHT** – switch the indicated light on (GREEN, RED, YELLOW) or all the lights off (OFF)
  SYNTAX: *label LIGHT - color 0 0 0*
  EXAMPLE: *10 LIGHT – GREEN 0 0 0 ; switch the green light on*
  RETURN: none

- **LOCALITY** – return the locality of the system
  SYNTAX: *label LOCALITY - - 0 0 0*
  EXAMPLE: *100 LOCALITY - - 0 0 0*
  RETURN: DESY, FNAL

- **MSG** – show a message in the status window
  SYNTAX: *label MSG - message 0 0 0*
  EXAMPLE: *100 MSG - Gate\sclosed 0 0 0*
  RETURN: none

- **NEXTCELL** – return the next cell to be tuned for the precise tuning method. Depending on the parameter the next cell can be returned as a string (STRING) or as an integer number (NUMBER).
  SYNTAX: *label NEXTCELL - {STRING|NUMBER} 0 0 0*
  EXAMPLE: *100 NEXTCELL - STRING 0 0 0*
  RETURN: STRING parameter: NONE, Cell #1 (Coupler), Cell #2, Cell #3, … , Cell #9
  NUMBER parameter: 0..9
- **NOOP** – no operation
  SYNTAX: *label NOOP - - 0 0 0*
  EXAMPLE: *10 NOOP - - 0 0 0 ; do nothing*
  RETURN: none

- **PRECISION** – set the tuning precision to either COARSE or FINE
  SYNTAX: *label PRECISION - precision 0 0 0*
  EXAMPLE: *100 PRECISION - FINE 0 0 0*
  RETURN: none

- **POPUP** – show a single button dialog window (a popup window) with a message. There are two special characters that can be included in the displayed text:
  - **\s** - a space character
  - **\n** – a newline character

  SYNTAX: *label POPUP - text 0 0 0*
  EXAMPLE: *0 POPUP - Reset\sinterlocks\nThen\scontinue 0 0 0*
  RETURN: none

- **PROMPT** – show a two-button dialog window (a popup window) with a message. The user has to press either the YES or NO button and the corresponding value is returned. The value can be used for conditional branching in the script. There are two special characters that can be included in the displayed text:
  - **\s** - a space character
  - **\n** – a newline character

  SYNTAX: *label PROMPT - text 0 0 0*
  EXAMPLE: *0 PROMPT - Do\syou\swant\sto\scontinue? 0 0 0*
  RETURN: YES, NO

- **RADIOPROMPT** – prompt the user with radio buttons to select the start of the script
  SYNTAX: *label RADIOPROMPT - - 0 0 0*
  EXAMPLE: *100 RADIOPROMPT - - 0 0 0*
  RETURN: none

- **REPLY** – get the latest reply from a given plug-in and put it in the return value
  SYNTAX: *label REPLY plug-in - 0 0 0*
  EXAMPLE: *10 REPLY gate - 0 0 0 ; get the reply from gate*
  RETURN: none

- **RETURN** – return from the subroutine and execute the next instruction after GOSUB
  SYNTAX: *label RETURN - - 0 0 0*
  EXAMPLE: *10 RETURN - - 0 0 0*
  RETURN: none (the last return value is unchanged)

- **RUN** – start plug-in (send the RUN command to plug-in) and flash the selected indicator
  SYNTAX: *label RUN plug-in - 0 indicator 0*
  EXAMPLE: *10 RUN gate - 0 1 0 ; run gate*
  RETURN: none

- **SAVE** – export data to a file in the DESY format
  SYNTAX: *label SAVE - X_01 0 0 0*
  EXAMPLE: *100 SAVE - X_01 0 0 0*
  RETURN: none

- **SCALE** – set the value of the script progress indicator (0 – 100)
  SYNTAX: *label SCALE - percent 0 0 0*
  EXAMPLE: *100 SCALE - 55 0 0 0*
  RETURN: none

- **SET** – set the plug-in's property value (typically used to set the RUN option/target.
  SYNTAX: *label SET plug-in property=value 0 0 0*
  EXAMPLE: *10 SET train TARGET=Home 0 0 0 ; move train to HOME*
  RETURN: none

- **SHOW** – show (switch view to) a given tab on the control screen
  SYNTAX: *label SHOW - tab 0 0 0*
  EXAMPLE: *10 SHOW - Gate 0 0 0*
  RETURN: none

- **SKIPENDCELLS** – modify the tuning plan to skip end cells
  SYNTAX: *label SKIPENDCELLS - - 0 0 0*
  EXAMPLE: *10 SKIPENDCELLS - - 0 0 0*
  RETURN: none

- **STATE** – get the current state of a given plug-in and put it in the return value
  SYNTAX: *label STATE plug-in - 0 0 0*
  EXAMPLE: *10 STATE gate - 0 0 0 ; get the state of gate*
  RETURN: none

- **TRAIN** – return the train status
  SYNTAX: *label TRAIN - - 0 0 0*
  EXAMPLE: *100 TRAIN - - 0 0 0*
  RETURN: UNKNOWN, MOVING, HOME, ECCENTRICITY,
  1 CELL @ GATE, 2 CELL @ GATE, ... , 9 CELL @ GATE,
  UNDEFINED, ERROR

- **TUNE** – check if tuning of the given cell is needed
  SYNTAX: *label TUNE - [1-9] 0 0 0*
  EXAMPLE: *100 TUNE - 2 0 0 0*
  RETURN: YES, NO

- **TUNEPLAN** – popup a tuning plan to be examined and adjusted if needed by the user
  SYNTAX: *label TUNEPLAN - - 0 0 0*
  EXAMPLE: *100 TUNEPLAN - - 0 0 0*
  RETURN: none

- **TYPE** – return the cavity type
  SYNTAX: *label TYPE - - 0 0 0*
  EXAMPLE: *100 TYPE - - 0 0 0*
  RETURN: cavity type name

- **WARNING** – log a warning
  SYNTAX: *label WARNING - warning_description 0 0 0*
  EXAMPLE: *9100 WARNING - Tuning\sscript\sdelayed 0 0 0*
  RETURN: none

- **WAIT** – wait until plug-in sends the specified reply or time-out (in seconds). The plug-in reply can be one of the following: *Error, Done, Done-NotReady, Initialized, Aborted, Aborted-NotReady, Exited.*
  SYNTAX: *label WAIT plug-in reply 0 indicator time-out*
  EXAMPLE: *10 WAIT gate Done 0 1 240 ; wait for gate to complete its run*
  RETURN: OK, ERROR, TIMEOUT

- **WAIT-ANY** – wait until plug-in sends any reply or time-out (in seconds). The plug-in reply can be one of the following: *Error, Done, Done-NotReady, Initialized, Running, Aborted, Aborted-NotReady, and Exited. The command's return value contains the received reply.*
  SYNTAX: *label WAIT-ANY plug-in - 0 indicator time-out*
  EXAMPLE: *10 WAIT-ANY tune - 0 -1 3000 ; wait for tune to reply*
  RETURN: TIMEOUT, *Done, Done-NotReady, Initialized, Running, Aborted, Aborted-NotReady, Exited*

3.14.8 Script Includes

Automation scripts may have common parts which can be included inside several scripts. A good example is a set of subroutines that can be invoked in several scripts. To facilitate this, the contents of another file can be included inside the script by using the include directive. The syntax of this directive is following:

#include file_name_to_include

Since there is no limit to the depth of includes, the included file may contain itself one or more include directives. The following example shows the use of the directive:

```
; ---------- SUBROUTINES -----------
#include subroutines.tas
```

3.14.9 Example script

Following is an example script with a subroutine and a mandatory abort handler.

```
; User script for the Cavity Tuning Machine
; Jerzy Nogiec, March 26, 2010
; vrs. 1.0

; ---------- SCRIPT ----------------------------------------------

; Display initial message and log it
0    SCALE   -    0                        0   0   0           ; set progress to 0%
0    MSG     -    User\sscript\sstarted    0   0   0           ; show message
0    INFO    -    User\sscript\sstarted    0   0   0           ; log information

; Prompt the user and continue until told to end
10     GOSUB    -         -             1000  0   0
0      BEQ      -    YES                 10     0   0          ; go to 10 if return value is YES

; End
0    POPUP   -   Ending\sscript                0   0   0       ; show a popup window
0    SCALE   -     100                         0   0   0       ; set progress to 100%
0    INFO    -  User\sscript\scompleted\sOK    0   0   0       ; log information
0    END     -     Script\sOK                  0   0   0       ; terminate the script OK

; ---------- SUBROUTINES  ------------------------------------

; Prompt the user and and pause
1000  PROMPT   -   Do\syou\swant\sto\scontinue?  0    0   0     ; prompt the user
0      DELAY    -      3                     0   0   0         ; pause for 3 seconds
0      RETURN  -      -                      0   0   0         ; return from subroutine

; ---------- ABORT -----------------------------------------------

; ABORT handler
9999 NOOP    -            -               0   0   0           ; do nothing
0   WARNING -    User\sscript\saborted    0   0   0           ; log a warning message
0   END      -      Aborted\sby\suser     0   0   0           ; terminate the script with an error
```

3.14.10 Exceptions

In order for the script to handle exceptions it must check the return codes from plug-ins and from subroutines and execute an appropriate sequence of recovery or termination instructions. In the following excerpt from a script the return code from a subroutine is checked and the operation is repeated, if failed.

```
0     MSG       -    Move\sto\sWIT             0    0  0          ; show message
100   GOSUB     -      WIT\sPosition          2000  0  0          ; move train to WIT position
0     BEQ       -           OK                110   0  0          ; go to 110 if move succeeded
0     PROMPT  -  Train\sfailed\nDo\syou\swant\sto\sretry? 0 0 0   ; prompt the user
0     BEQ       -         YES                 100   0  0          ; retry if requested by the user
0     JMP        -        -                   900   0  0          ; go to error handler otherwise ----->
110   NOOP      -        -                     0    0  0          ; continue script
```

3.14.11 Set Command

The Set command is used to modify properties in plug-ins. It has the following format:

property_name = property_value

Each plug-in can have a different property or properties and expects only allowed property values. Below are included property names and allowed values for several plug-ins.

Plug-in	*property*	*Allowed values*
Spectrum	Msr	Sub Modes Pi-Mode
Gate	Msr	Open Close
Train	Target	Home WIT Position Eccentricity Cell #1 (Coupler) Cell #2 Cell #3 Cell #4 Cell #5 Cell #6 Cell #7 Cell #8 Cell #9
Bead-pull	Run	Move to Home, Move to Home Train Move to Home Fixed Fast Data, Fast Data Train Fast Data Fixed Move to Park, Move to Park Train Precision data, Precision data Train Precision data Fixed Amplitude Data, Amplitude Data Train Amplitude Data Fixed
Eccentricity Model	Model	Eccentricity Perpendicularity
Eccentricity	Run	Measurement MoveToPark
Eccentricity	Mode	Continuous Mode Step Mode
Model	Model	Multi-mode PI-mode Auto